Name: _____ **Section:** 1 2 3 4

1 = Stouder, 2nd–3rd periods. **2 =** Stouder, 4th–5th periods. **3 =** Mutchler, 7th-8th periods. **4 =** Rupakheti, 9th-10th periods.

Use this quiz to help make sure you understand the videos/reading. ***Answer all questions.*** Make additional notes as desired. ***Not sure of an answer?*** Ask your instructor to explain in class and revise as needed then. ***Please print two-sided if practical.***

Throughout, where you are asked to "circle your choice", you can circle or underline it (whichever you prefer).

Video: ***Sequences*** [8:43 minutes]

1. Sequences are powerful because:

2. What gets printed by the program shown below when *main* runs?

```
def main():
    list1 = [74, 34, 13, 30, 4004]
    string1 = 'Expert texpert!'

    examples_of_indexing(list1)
    examples_of_indexing(string1)

def examples_of_indexing(sequence):
    length = len(sequence)
    print('Length is', length)

    print('At indices 0, 1, 4:')
    print(sequence[0])
    print(sequence[1])
    print(sequence[4])

    print('List them all:')
    for k in range(len(sequence)):
        print(sequence[k])
    print()

    print('Cool stuff:')
    print(sequence[-1])

    print('Last item:')
    length = len(sequence)
    print(sequence[length - 1])
```

Just take a wild guess and ask in class!

Output
(fill in the blanks)

```
Length is _______

At indices 0, 1, 4:

_____________

_____________

_____________

List them all:

_____________

_____________

_____________

_____________

_____________


Cool stuff:

_____________

Last item:

_____________
```

(output continues, from second call to *examples_of_indexing*)

```
Length is _______

At indices 0, 1, 4:

_____________

_____________

_____________

List them all:

_____________

_____________

_____________
```

...[don't bother doing all of the letters, just skip to the last one] _____

```
Cool stuff:

_____________

Last item:

_____________
```

3. Implement (here, on paper) the following function, per its specification.

```
def count_negative(seq):
    """
    Returns the number of items in the given sequence of numbers
    that are negative.

    Precondition: The argument is a sequence of numbers.
    """
```

4. Implement (here, on paper) the following function, per its specification.

```
def count_short_ones(seq_of_lists):
    """
    The argument is a sequence of lists.  Returns the number of
    items in that sequence whose length is less than 3.

    For example, if the argument is:
        [ [3, 5], [3, 9, 0, 4], [5], [5], [], [9, 8, 7], [5, 6] ]
    then this function returns 5, since 5 of the 7 lists in the
    above sequence have length less than 3
    (I have displayed those 5 lists in red.)

    Precondition: The argument is a sequence of lists.
    """
```

Video: ***The Last Item in a Sequence*** [3:20 minutes]

5. Given a sequence called ***seq*** that contains `10` items, write expressions for:

   - The beginning item (element) in ***seq***: _____
   - The last item (element) in ***seq***: _____

6. Suppose that you have a ***list*** called ***my_list*** that contains `10` items. What error message appears if you attempt to access ***my_list*****[10]**?

7. Suppose that you have a ***string*** called ***my_string*** that contains `10` characters. What error message appears if you attempt to access ***my_string*****[10]**?

8. In Eclipse, when a run-time *exception* (error) occurs, how do you go straight to the line at which the exception occurred?

9. Is that line always the line that is wrong? **Yes** **No** (circle your choice)

10. What should you do if the exception occurred in the *zellegraphics* or *new_create* modules?

11. In the two code fragments below, assume that the variable **s** has been assigned as its value some sequence. The code on the left (below) causes a run-time exception. The code on the right (below) does NOT cause a run-time exception. But both use the same expression **len(s)**. Explain why the code on the left breaks (and is wrong) but the code on the right does not break (and is correct).

```
... s[len(s)] ...
```

```
for k in range(len(s)):
    ... s[k] ...
```

Video: ***Iterating Through a Sequence*** [11:35 minutes]

In each of the following problems, assume that there is a variable **x** that is a sequence of numbers.

12. Write a loop that prints all the elements of **x**, backwards. For example, if **x** were the list **[9, 4, 12, 88, 17],** then your loop should print **17**, **88**, **12**, **4** and **9**, in that order.

13. Write a loop that prints all the elements of **x** between indices **4** and **7**, backwards. For example, if **x** were the list **[9, 4, 12, 88, 17, 33, 20, 21, 22]**, then your loop should print **21**, **20**, **33** and **17**, in that order. Assume that the sequence has at least **8** elements.

14. Write a loop that prints every 3rd element of **x,** starting at index **1**. For example, if **x** were the list **[9, 4, 12, 88, 17, 33, 20, 21, 22, 5, 7, 77]**, then your loop should print **4**, **17**, **21** and **7**, in that order. Assume that the sequence has at least **2** elements.

15. Write a loop that prints all the elements of **x** that are greater than **10**. For example, if **x** were the list **[9, 4, 12, 88, 17, 33, 20, 21, 22, 5, 7, 77]**, then your loop should print **12**, **88**, **17**, **33**, **20**, **21**, **22** and **77**, in that order.

Textbook Reading: ***Section 6.1 — Basic Properties of Lists*** (pages 278 - 283)

16. Write a statement that constructs a ***list*** containing three numbers (any three, you pick them, whatever you want) and assigns the variable ***`numbers`*** to that list.

17. Continuing the previous problem, write the statement that changes the ***last*** element in the list to which the variable ***`numbers`*** refers, to **66**. (So after this statement executes, the last element in the list will be **66**, instead of whatever you set it to in the previous problem.)

18. Consider Figure 1 on page 279, as shown below for your convenience:

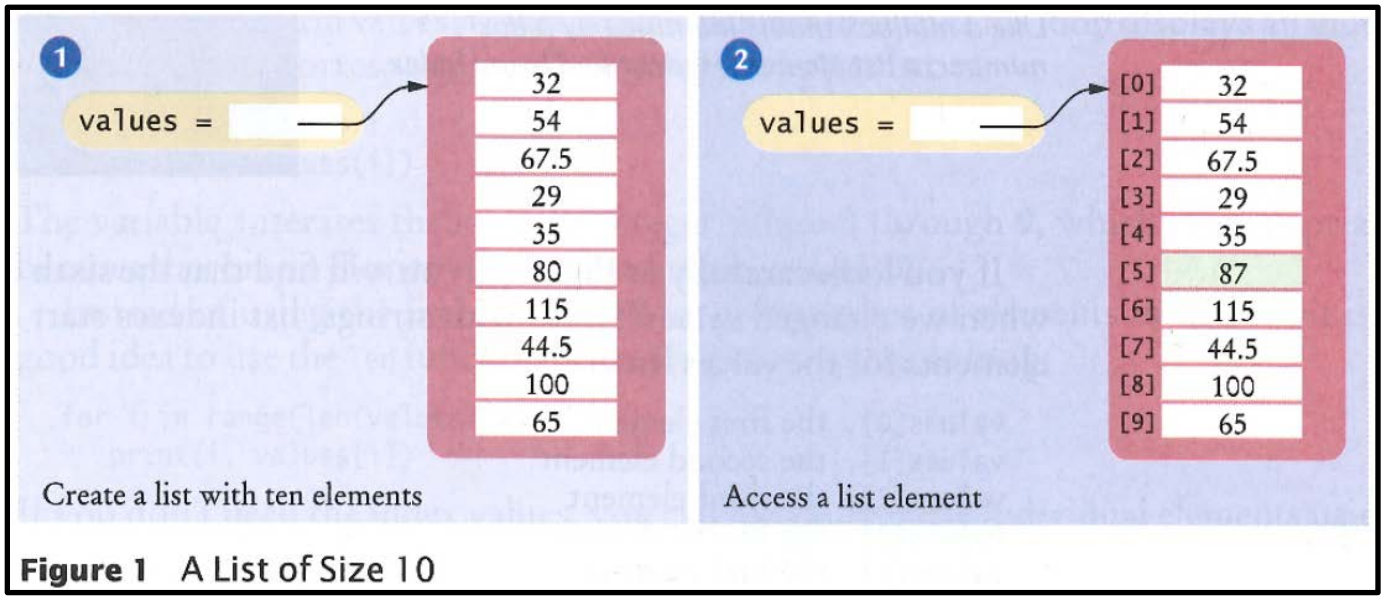


Figure 1 A List of Size 10

The left-hand-side of the figure shows the result of the statement:

```
values = [32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65]
```

The right-hand-side of the figure shows the result of the additional statement:

```
values[5] = 87
```

Write a statement that modifies the right-hand-side of the picture to make the element which is **`100`** in the picture become **`-33.33`**.

19. Suppose that the value **x** refers to a list. Write a statement that prints the number of elements in the list.

20. What is the difference between the following two expressions?

```
numbers[3]
```

```
numbers = [3]
```

21. Suppose that the value **numbers** refers to a list of numbers. Write a loop that traverses (i.e., goes through) the list, making each number in the list one bigger than it was.

22. [Self Check problem 1 from page 282]. Define a variable called ***primes*** that refers to a list that contains the first five prime numbers (i.e., 2, 3, etc).

23. [Self Check problem 2 from page 282]. Continuing the previous problem, what does the list contain after executing the following loop?

```
for m in range(5):
    primes[m] = primes[m] + 1
```

24. [Self Check problem 5 from page 282]. Define a variable (whose name can be whatever you choose for it) that refers to a list containing two strings, **'Yes'** and **'No'**.

25. What does the following code snippet print, and what happens when the last of the statements executes? (Write the output directly to the right of the ***print*** statements.)

```
names = ['zoran', 'siti', 'elchin']

print(names[0])

print(names[2])

print(names[0][0])

print(names[2][4])

print(names[3])
```

26. Read **Computing & Society 6.1 *Computer Viruses***, on page 283. (For your convenience, I have pasted a copy of it on the next page of this quiz.) After reading it:

    a. In November 1988, Robert Morris launched a computer virus (actually, a form of virus called a ***worm***) that, through an error in his code, delivered a denial-of-service attack to computers on the Internet. He was convicted under the *The Computer Fraud and Abuse Act*. After appeal, what was his sentence?

    _____ years probation

    _____ hours of community service

    _____ fine

    b. His worm worked by attacking a program called ***finger*** that was written in the programming language C. If ***finger*** had been written (and run) in ***Python***, would his attack have succeeded? Why or why not?

    c. Look on the Internet for more about the Morris' attack. (A Google search reveals plenty.) After reading a bit, do you think that his sentence was (circle your choice):

Much too lenient	**Somewhat too lenient**	**About right**	**Somewhat too harsh**	**Much too harsh**

    d. Robert Morris' father is famous. What are some of the things for which he is famous? Which one of those things is particularly ironic given his son's behavior?

Computing & Society 6.1 Computer Viruses

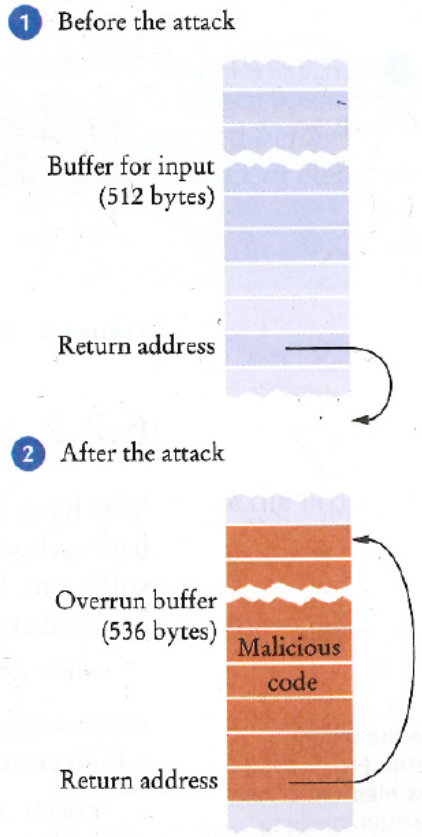
In November 1988, Robert Morris, a student at Cornell University, launched a so-called virus program that infected about 6,000 computers connected to the Internet across the United States. Tens of thousands of computer users were unable to read their e-mail or otherwise use their computers. All major universities and many high-tech companies were affected. (The Internet was much smaller then than it is now.)

The particular kind of virus used in this attack is called a *worm*. The worm program crawled from one computer on the Internet to the next. The worm would attempt to connect to finger, a program in the UNIX operating system for finding information on a user who has an account on a particular computer on the network. Like many programs in UNIX, finger was written in the C language. In order to store the user name, the finger program allocated an array of 512 characters (an array is a sequence structure similar to a list), under the assumption that nobody would ever provide such a long input. Unfortunately, C does not check that an array index is less than the length of the array. If you write into an array using an index that is too large, you simply overwrite memory locations that belong to some other objects. In some versions of the finger program, the programmer had been lazy and had not checked whether the array holding the input characters was large enough to hold the input. So the worm program purposefully filled the 512-character array with 536 bytes. The excess 24 bytes would overwrite a return address, which the attacker knew was stored just after the array. When that method was finished, it didn't return to its caller but to code supplied by the worm (see the figure, A "Buffer Overrun" Attack). That code ran under the same super-user privileges as finger, allowing the worm to gain entry into the remote system. Had the programmer who wrote finger been more conscientious, this particular attack would not be possible.

In Python, as in C, all programmers must be very careful not to overrun the boundaries of a sequence. However, in Python, this error causes a run-time exception and never corrupts memory outside the list.

One may well speculate what would possess the virus author to spend many weeks to plan the antisocial act of breaking into thousands of computers and disabling them. It appears that the break-in was fully intended by the author, but the disabling of the computers was a bug, caused by continuous reinfection. Morris was sentenced to 3 years probation, 400 hours of community service, and a $10,000 fine.

In recent years, computer attacks have intensified and the motives have become more sinister. Instead of disabling computers, viruses often steal financial data or use the attacked computers for sending spam e-mail. Sadly, many of these attacks continue to be possible because of poorly written programs that are susceptible to buffer overrun errors.



A "Buffer Overrun" Attack